REPORT

COURSE : OPERATING SYSTEMS

Assignment - Simple Operating System

APRIL 8TH TO MAY 30TH 2023, HO CHI MINH CITY, VIETNAM

AGRI TERRA

HCMC University of Technology

Block A3, 268 Ly Thuong Kiet
Ward 14, District 10
HCM City
T +84 28 3864 7256

Faculty of Computer Science and Engineering,
Department of Systems and Networkings

Operating System

Simple Operating System

Assignment Report

Author(s) :
Title : Simple Operating System
Publisher : Agriterra
Project Number :
Number AgriStudies : [number]
Country : [countryname]
Category : [category]

CONTENTS

SUMMARY

Bài tập lớn tập trung vào thiết kế một hệ điều hành đơn giản dựa trên hai thành phần chính :

+ **Scheduler** : là giải thuật định thời giúp xác định process được thực hiện trên CPU chỉ định
+ **Memory Management** : phân biệt các không gian bộ nhớ thực của các process với nhau bằng cách mỗi process sở hữu một không gian bộ nhớ ảo riêng biệt và hệ thống sẽ sắp xếp và truyền địa chỉ trong bộ nhớ ảo của process đến địa chỉ trong bộ nhớ thực tương ứng.

Thông qua bài tập lớn, sinh viên có thể hiểu được hơn về nguồn gốc và nguyên lý hoạt động của một hệ điều hành đơn giản, tập trung về các nội dung Scheduling, Memory Management và Synchronization.

Bằng những kiến thức tìm hiểu được thông qua bài tập lớn và các bài giảng, nhóm đã thành công thiết kế được một hệ điều hành đơn giản sẽ được giới thiệu chi tiết hơn trong các nội dung sau.

Checklist	Y/N
Is it ½ page maximum	
Is it written in an active voice?	
Does it answer Who, What, When, Where, Why and How?	
Does the opening paragraph describe project objectives and its purpose?	
Does it present contextual background and principle (in short aka less than 1 page)?	
Does it highlight the report's final finding or project contributions or results?	
Does the closing paragraph provide a conclusion (optional: next step recommendation)?	

Is it consistent in the usage of the same keywords, terms, technical phrases?	
Have the authors check and eliminate the statement/information that is not discussed in the main body of the report	

Author(s)

1 INTRODUCTION TO THE ASSIGNMENT

1.1 Introduction

Khi công nghệ ngày càng phát triển, lượng thông tin và tác vụ cần xử lý ngày càng lớn và phức tạp hơn, dẫn đến phần cứng không thể đáp ứng được các tác vụ ấy. Do đó, hệ điều hành được thiết kế để giải quyết các vấn đề về sắp xếp và quản lý tài nguyên và tác vụ.

Trong bài tập lớn này, ta sẽ nghiên cứu về hai thành phần chính để hiện thực một hệ điều hành đơn giản, bao gồm Scheduler và Paging-based Memory Management.

1.1.1 Scheduler

Multi-level queue scheduling là một cấu trúc giải thuật để sắp xếp các tác vụ dựa trên mức độ ưu tiên của các tác vụ ấy và cung cấp tài nguyên tương ứng.

Để hiện thực được scheduler, sinh viên phải có kiến thức về các thành phần liên quan đến Multi-level queue scheduling gồm :

+ Hiện thực queue (enqueue / dequeue)
+ Priority và số slot trong queue dựa trên priority

Thông qua bài tập lớn này, sinh viên sẽ hiểu rõ hơn về nguyên lý hoạt động của giải thuật này và tác dụng của nó trong Hệ điều hành.

1.1.2 Paging-based Memory Management

Vì bộ nhớ thực là bộ nhớ hữu hạn nên bộ nhớ ảo được sinh ra để tăng dung lượng tác vụ có thể xử lý và hiệu suất của hệ điều hành.

Để hiện thực được memory management, sinh viên phải có kiến thức về các thành phần trong bộ nhớ gồm :

+ Bộ nhớ ảo
+ Bộ nhớ thực
+ Ánh xạ giữa hai bộ nhớ dựa trên kỹ thuật phân trang

Thông qua bài tập lớn này, sinh viên sẽ hiểu rõ hơn về nguyên lý hoạt động của Memory Management Unit(MMU) khi xử lý bộ nhớ thông qua bảng phân trang.

1.1.3 Synchronization

Khi tài nguyên được sử dụng bởi nhiều task sẽ dẫn đến race condition, do đó cần được bảo vệ nhằm tránh xảy ra sai lệch dữ liệu.

Để hiện thực Synchronization, sinh viên phải có kiến thức về tài nguyên cần được bảo vệ và cách bảo vệ tài nguyên khi hiện thực.

Thông qua bài tập lớn này, sinh viên sẽ có nghiên cứu sâu hơn về race condition và cách để tránh xảy ra race condition.

CHECKLIST

Checklists	Y/N
Why was the work done?	

What must be known to understand the rest of the report?	
Explain the historical issue behind the research and its significance? Did a specific issue/event impact this study? Is it a subsequent phase of R&D study? (YES- it illustrate class theory) Is it breaking new ground? (Usually assignment NO)	

2 FINDINGS AND CONCLUSIONS

2.1 Scheduler

Một trong hai thành phần quan trọng khi hiện thực một hệ điều hành đơn giản là **Scheduler** giúp hệ thống sắp xếp các process theo thứ tự ưu tiên để tăng độ tương tác giữa hệ thống và người dùng. Trong bài tập lớn này, sinh viên cần hiểu về cách hoạt động của giải thuật **Multi-level queue scheduling** và **Round Robin** và hiện thực các hàm **dequeue**, **enqueue** và **get_mlq_proc**. Thông qua **Scheduler**, các process sẽ được chạy luân phiên dựa trên mức độ ưu tiên, giúp hệ thống tối ưu được hiệu năng và tương tác. Ở phần này, nhóm đã hiện thực và thiết kế **Scheduler** có khả năng sắp xếp và phân phối thời gian cho các process. Sau khi thử nghiệm thì nhóm quan sát được rằng tính năng đã được hiện thực thành công. Sau khi hoàn thành phần này, sinh viên sẽ hiểu được nguyên lý hoạt động và tác dụng của **Scheduler** và các giải thuật định thời trong một hệ điều hành đơn giản.

2.2 Paging-based Memory Management

Thành phần quan trọng còn lại là **Paging-based memory management** được hiện thực dựa trên kỹ thuật phân trang (Paging-based). Với kỹ thuật này, bộ nhớ ảo sẽ được chia thành các **page** còn bộ nhớ thực sẽ chia thành các **frame** gồm **RAM** và **SWAP** để hiện thực swapping. Trong bài tập lớn này, sinh viên cần hiểu được các thông số cần thiết của một hệ điều hành cơ bản như RAM, SWAP và kỹ thuật phân trang và hiện thực các hàm **__free**, **vmap_page_range**, **pg_getpage**, **__alloc**, **alloc_pages_range** và **MEMPHY_dump**. Thông qua **Paging-based Memory Management**, hệ thống có thể xử lý các process có kích thước lớn chỉ với một bộ nhớ thực hữu hạn. Với phần này, nhóm đã hiện thực và thiết kế hệ thống giúp quản lý và cấp phát tài nguyên để thực thi chương trình. Sau khi thực nghiệm thì nhóm quan sát được rằng tính năng này đã được hiện thực thành công. Sau khi hoàn thành phần này, sinh viên sẽ hiểu được nguyên lý hoạt động của kỹ thuật phân trang và swapping khi xử lý bộ nhớ thực và bộ nhớ ảo trong một hệ điều hành đơn giản.

2.3 Conclusions

Nhóm đã hiện thực thành công mô phỏng một hệ điều hành đơn giản, sử dụng 2 chức năng là **Multi-level queue scheduling** với CPU scheduler là **Round Robin**, các hàm xử lý **Paging Memory Management**. Cùng với đó là đã thí nghiệm thành công các test case để đánh giá độ chính xác của giải thuật (sẽ được giới thiệu ở phần 3).

CHECKLIST

Checklists	Y/N
Provides statements (keep it (max) of 5 bullet items)	
What project should we achieve? Verify the consistency these statement with evaluations do later	

What was the work conducted for the project? Minus listing the work conducted. Major task completed for the projects	

3 ACTIONS FOR FOLLOW-UP

3.1 Module 1 - Scheduler

3.1.1 Design

Hệ điều hành được thiết kế để thực thi nhiều tác vụ. Bằng cách sử dụng nhiều hàng đợi gọi là **ready_queue** để xác định process được thực thi khi có CPU trống. Mỗi hàng đợi sẽ có độ ưu tiên được định nghĩa trước. Scheduler được thiết kế dựa trên giải thuật "multilevel queue" được sử dụng trong Linux nhằm thể hiện độ ưu tiên của các process kết hợp với giải thuật định thời round robin để tăng độ tương tác với người dùng.

Khi bắt đầu thực thi chương trình, hệ điều hành sẽ load các chương trình (program) và gắn PCB cho từng program. Sau đó, hệ thống sẽ đọc và lưu các lệnh của program vào PCB, và đưa PCB của process vào **ready_queue** tương ứng với **prio** của process. CPU sẽ thực thi các process theo thuật toán round-robin với thời gian cho một process là **time slice** được định nghĩa trước. Khi process được thực thi xong hoặc hết thời gian cho process, process sẽ được đưa lại vào **ready_queue** và CPU sẽ lấy process khác từ **ready_queue** để thực thi.

3.1.2 Implementation

- **enqueue** :

```
10 void enqueue(struct queue_t * q, struct pcb_t* proc)
11 {
12         /* TODO: put a new process to queue [q] */
13         if (q && proc)
14         {
15                 if (!q->size)
16                 {
17                         if (!q->head)
18                                 q->head = (struct pcbList*)malloc(sizeof(struct pcbList));
19
20                         q->head->proc = proc;
21                         q->head->next = q->head->prev = NULL;
22                         q->tail = q->head;
23                 }
24                 else
25                 {
26                         struct pcbList* temp = (struct pcbList*)malloc(sizeof(struct pcbList));
27                         temp->proc = proc;
28                         temp->next = temp->prev = NULL;
29
30                         q->tail->next = temp;
31                         temp->prev = q->tail;
32                         q->tail = temp;
33                 }
34                 q->size++;
35         }
36 }
37
```

- **dequeue** :

```
38 struct pcb_t * dequeue(struct queue_t * q)
39 {
40         /* TODO: return a pcb whose prioprity is the highest
41          * in the queue [q] and remember to remove it from q
42          * */
43         if (q && !empty(q) && q->head)
44         {
45                 struct pcb_t* result = q->head->proc;
46
47                 if (q->head->next)
48                 {
49                         q->head = q->head->next;
50                         free(q->head->prev);
51                         q->head->prev = NULL;
52                 }
53                 else
54                 {
55                         free(q->head);
56                         q->head = q->tail = NULL;
57                 }
58
59                 q->size--;
60                 //q->run++;
61
62                 return result;
63         }
64
65         /*if (empty(q))
66                 q->run = 0;*/
67
68         return NULL;
69 }
```


- **get_mlq_proc** :

```c
struct pcb_t * get_mlq_proc(void)
{
        /*TODO: get a process from PRIORITY [ready_queue].
         * Remember to use lock to protect the queue.
         * */

        pthread_mutex_lock(&queue_lock);
        struct pcb_t * proc = NULL;
        int isEmpty = 0; // value for the second check, if mlq stills empty, then return null
        for (int i = currPrio; i < MAX_PRIO; i++)
        {
                currPrio = i;
                if (mlq_ready_queue[i].size > 0)
                {
                        mlq_ready_queue[i].run += time_slot;

                        //if (mlq_ready_queue[i].run <= mlq_ready_queue[i].maxRun)
                        proc = dequeue(&mlq_ready_queue[i]);

                        if (mlq_ready_queue[i].run >= mlq_ready_queue[i].maxRun || empty(&mlq_ready_queue[i]))
                        {
                                mlq_ready_queue[i].run = 0;
                                currPrio++;
                                if (currPrio >= MAX_PRIO - 1)
                                        currPrio = 0;
                        }

                        break;
                }

                if (currPrio >= MAX_PRIO - 1)
                {
                        currPrio = 0;

                        if (isEmpty)
                                break;

                        i = -1;
                        isEmpty = 1;
                }
        }
        pthread_mutex_unlock(&queue_lock);
        return proc;
}
```

3.1.3 Test

- Input : test_sched

```
4 2 4
0 s0 0
1 s1 2
2 s2 1
5 s3 3
```

-

Time slice = 4, Number of CPUs = 2, Number of process = 4

+ Process :

s0	s1	s2	s3
12 15 calc calc calc calc calc calc calc calc calc calc	20 7 calc calc calc calc calc calc calc	20 12 calc calc calc calc calc calc calc calc calc calc	7 11 calc calc calc calc calc calc calc calc calc calc

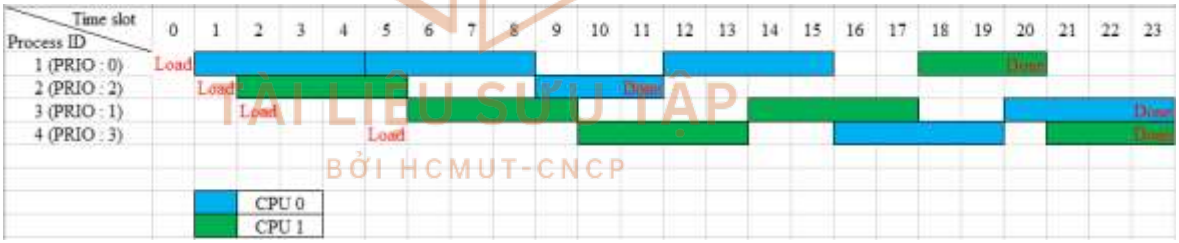
calc calc calc calc calc		calc calc calc	calc calc calc calc calc calc calc

+ Output : ./os test_sched

```
Time slot   0
ld_routine
        Loaded a process at input/proc/s0, PID: 1 PRIO: 0
Time slot   1
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/s1, PID: 2 PRIO: 2
Time slot   2
        CPU 1: Dispatched process  2
        Loaded a process at input/proc/s2, PID: 3 PRIO: 1
Time slot   3
Time slot   4
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/s3, PID: 4 PRIO: 3
Time slot   6
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  3
Time slot   7
Time slot   8
Time slot   9
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  10
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  4
```

```
Time slot   11
Time slot   12
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot   13
Time slot   14
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  3
Time slot   15
Time slot   16
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot   17
Time slot   18
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  1
Time slot   19
Time slot   20
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot   21
        CPU 1: Processed  1 has finished
        CPU 1: Dispatched process  4
Time slot   22
Time slot   23
Time slot   24
        CPU 1: Processed  4 has finished
        CPU 1 stopped
        CPU 0: Processed  3 has finished
        CPU 0 stopped
```

+ Gantt chart :



3.2 Module 2 - Paging-based memory management

3.2.1 Design

Khi một process được thực thi bởi CPU, process đó sẽ được cấp phát một vùng nhớ trên bộ nhớ thực. Theo cơ chế paging, bộ nhớ vật lý được chia thành các **frame** bằng nhau và bộ nhớ luận lý được chia thành các **page** bằng nhau ánh xạ với nhau để cung cấp vùng nhớ cho process.

3.2.1.1 Virtual memory

Trong mỗi process sẽ có một con trỏ trỏ đến địa chỉ của **memory map** chứa các **virtual memory area**. Các **virtual memory area** sẽ được chia thành các trang với kích thước bằng với kích thước của 1 frame trong RAM và được ánh xạ tới các frame trong RAM, các **virtual memory region** là các vùng nhớ ảo được sử dụng trong **virtual memory area** (có thể không liền kề với nhau và có kích thước khác nhau).

Trong không gian bộ nhớ ảo, địa chỉ được chia thành hai phần :

+ **Page number** : là chỉ số trang, dùng để ánh xạ đến **frame** tương ứng trong bộ nhớ thực.
+ **Page offset** : là vị trí của địa chỉ trong trang, khi kết hợp với **Page number** sẽ tạo ra địa chỉ vật lý.

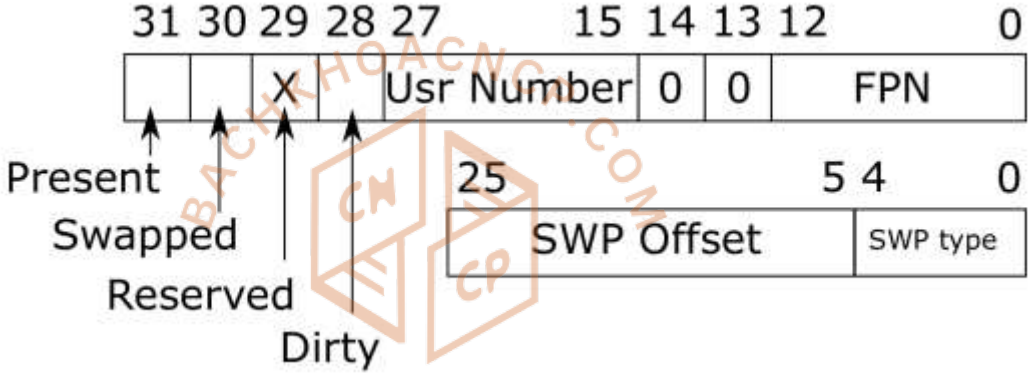
3.2.1.2 Physical memory

Bộ nhớ vật lý gồm hai phần : RAM và SWAP. Các process muốn được thực thi cần phải được load lên RAM. Trong kỹ thuật phân trang, RAM được chia thành các frames bằng nhau về kích thước. Tương tự, bộ nhớ vật lý SWAP cũng được chia thành các frames.

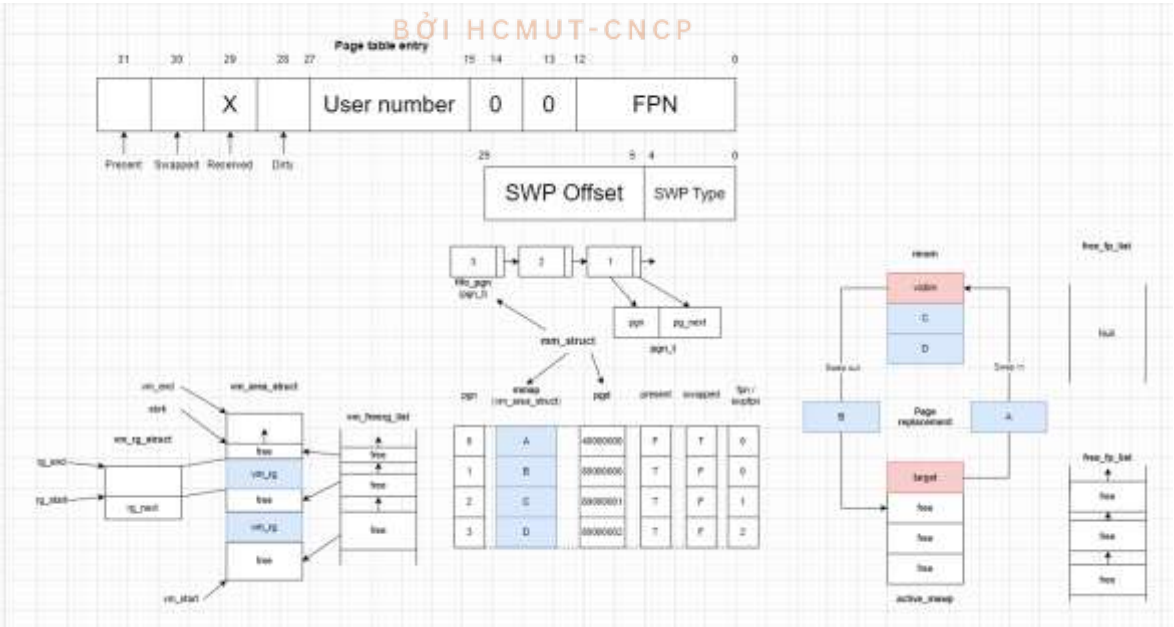
Trong thực tế, bộ nhớ RAM là hữu hạn nên không đủ lớn cho mọi process. Do đó, cơ chế swapping được sử dụng để chuyển frame giữa RAM và SWAP, giúp cho hệ thống thực thi các chương trình có kích thước lớn hơn RAM.

3.2.1.3 Paging-based address translation scheme

Mỗi process đều có một bảng phân trang để thực hiện ánh xạ các page của nó tới các frame tương ứng trong bộ nhớ vật lý.



3.2.1.4 Diagram



3.2.2 Implementation

3.2.2.1 mm-vm.c

```
int __alloc(struct pcb_t *caller, int vmaid, int rgid, int size, int *alloc_addr)
{
  /*Allocate at the toproof */
  if (caller->mm->symrgtbl[rgid].rg_start < caller->mm->symrgtbl[rgid].rg_end)
    pgfree_data(caller, rgid);

  struct vm_rg_struct rgnode;

  if (get_free_vmrg_area(caller, vmaid, size, &rgnode) == 0)
  {
    caller->mm->symrgtbl[rgid].rg_start = rgnode.rg_start;
    caller->mm->symrgtbl[rgid].rg_end = rgnode.rg_end;

    *alloc_addr = rgnode.rg_start;
```

+ __alloc :

```
    return 0;
  }

  /* TODO get_free_vmrg_area FAILED handle the region management (Fig.6)*/

  /*Attempt to increate limit to get space */
  struct vm_area_struct *cur_vma = get_vma_by_num(caller->mm, vmaid);
  int inc_sz = PAGING_PAGE_ALIGNSZ(size);
  //int inc_limit_ret
  int old_sbrk;

  old_sbrk = cur_vma->sbrk;

  /* TODO INCREASE THE LIMIT
   * inc_vma_limit(caller, vmaid, inc_sz)
   */

  if (old_sbrk + size > cur_vma->vm_end)
  {
     if (inc_vma_limit(caller, vmaid, inc_sz) < 0)
     {
        printf("Cannot increase the vm limit!\n");
        return -1;
     }
  }
  cur_vma->sbrk += size;

  /*Successful increase limit */
  caller->mm->symrgtbl[rgid].rg_start = old_sbrk;
  caller->mm->symrgtbl[rgid].rg_end = old_sbrk + size;

  *alloc_addr = old_sbrk;


  return 0;
}
```

+ __free :

```
int __free(struct pcb_t *caller, int vmaid, int rgid)
{
  struct vm_rg_struct* rgnode;

  if(rgid < 0 || rgid > PAGING_MAX_SYMTBL_SZ)
    return -1;

  /* TODO: Manage the collect freed region to freerg_list */
  rgnode = (get_symrg_byid(caller->mm,rgid));
  /*enlist the obsoleted memory region */
  enlist_vm_freerg_list(caller->mm, rgnode);
  rgnode->rg_start = rgnode->rg_end = -1;
  rgnode->rg_next = NULL;
  return 0;
}
```

+ pg_getpage :

```
int pg_getpage(struct mm_struct *mm, int pgn, int *fpn, struct pcb_t *caller)
{
  uint32_t pte = mm->pgd[pgn];

  if (!PAGING_PAGE_PRESENT(pte))
  { /* Page is not online, make it actively living */
    int vicpgn, swpfpn;
    int vicfpn;
    uint32_t vicpte;

    //int tgtfpn = PAGING_SWP(pte);
    int tgtfpn = PAGING_SWPOFF(pte); //the target frame storing our variable in swap memory
    int tgtswp_type = PAGING_SWPTYP(pte);

    /* TODO: Play with your paging theory here */
    /* Find victim page */
    if (find_victim_page(caller->mm, &vicpgn) < 0)
        return -1;

    while(vicpgn == pgn)
    {
        if (find_victim_page(caller->mm, &vicpgn) < 0)
           return -1;
    }

    vicpte = caller->mm->pgd[vicpgn];
    vicfpn = PAGING_FPN(vicpte);

    /* idx for mswp*/
    int i = 0;

    struct memphy_struct* mswp = (struct memphy_struct*)caller->mswp;
    for (i = 0; i < PAGING_MAX_MMSWP; i++)
    {
       if (mswp + i == caller->active_mswp)
         break;
    }
```

```
    /* Get free frame in MEMSWP */
    if(MEMPHY_get_freefp(caller->active_mswp, &swpfpn) < 0){
      struct memphy_struct** mswpit = caller->mswp;
      for(i = 0; i < PAGING_MAX_MMSWP;i++){
        struct memphy_struct* tmp_swp = (struct memphy_struct*)(mswpit);
        if(MEMPHY_get_freefp(tmp_swp+i,&swpfpn) == 0){
          /* Do swap frame from MEMRAM to MEMSWP and vice versa*/
          /* Copy victim frame to new swap frame  */
          __swap_cp_page(caller->mram,vicfpn,tmp_swp+i,swpfpn);
          caller->active_mswp = tmp_swp+i;
          break;
        }
      }
      /* dump */
      /*printf("Out of Memory !!\n");
      return -1;*/
    }
    else
      __swap_cp_page(caller->mram,vicfpn,caller->active_mswp,swpfpn);

    /* Copy target frame from swap to vicfpn in RAM */
    __swap_cp_page(mswp + tgtswp_type, tgtfpn, caller->mram, vicfpn);
    //__swap_cp_page(caller->active_mswp,tgtfpn,caller->mram,vicfpn);

    /*free frame in swap that includes tgtfpn which have been copy to RAM*/
    MEMPHY_put_freefp(mswp + tgtswp_type,tgtfpn);

    /* Update page table, set swap for idx vicpgn  */
    pte_set_swap(&caller->mm->pgd[vicpgn],i,swpfpn);

    /* Update its online status of the target page */
    //set online frame for idx pgn with the vicfpn which has move to swap
    pte_set_fpn(&caller->mm->pgd[pgn], vicfpn);

    // keep tracking
    enlist_pgn_node(&caller->mm->fifo_pgn,pgn);
    pte = caller->mm->pgd[pgn];
  }

  *fpn = PAGING_FPN(pte);

  return 0;
}
```

+ validate_overlap_vm_area :

```
int validate_overlap_vm_area(struct pcb_t *caller, int vmaid, int vmastart, int vmaend)
{
  struct vm_area_struct *vma = caller->mm->mmap;

  /* TODO validate the planned memory area is not overlapped */
  while(vma != NULL)
  {
      if (vmaid != vma->vm_id)
      {
         if (OVERLAP(vmastart,vmaend,vma->vm_start,vma->vm_end))
         {
             if ((vmastart != vma->vm_end || vmaend == vma->vm_start) &&
                (vmastart == vma->vm_end || vmaend != vma->vm_start))
                 return -1;
         }
      }
      vma = vma->vm_next;
  }

  return 0;
}
```

+ find_victim_page :

```
int find_victim_page(struct mm_struct *mm, int *retpgn)
{
  struct pgn_t *pg = mm->fifo_pgn;

  if (!pg)
     return -1;

  /* TODO: Implement the theorical mechanism to find the victim page */
  while (pg->pg_next && pg->pg_next->pg_next)
     pg = pg->pg_next;

  if (pg->pg_next)
  {
     *retpgn = pg->pg_next->pgn;
     free(pg->pg_next);
     pg->pg_next = NULL;
  }
  else
  {
     *retpgn = pg->pgn;
     free(pg);
     mm->fifo_pgn = pg = NULL;
  }

  return 0;
}
```


3.2.2.2 mm-memphy.c

+ MEMPHY_dump :

```c
int MEMPHY_dump(struct memphy_struct * mp)
{
    /*TODO dump memphy contnt mp->storage
     *     for tracing the memory content
     */
    char result[100];
    strcpy(result, "Memory content-[pos, content]: ");
    char temp[100];
    if (mp && mp->storage)
    {
        for (int i = 0; i < mp->maxsz; i++)
        {
            if (mp->storage[i] != (char)0)
            {
                sprintf(temp, "[%d, %d]", i, mp->storage[i]);
                strcat(result, temp);
            }
        }
        strcat(result, "\n\0");
    }
    printf("%s", result);
    return 0;
}
```

3.2.2.3 mm.c

+ vmap_page_range :

```c
int vmap_page_range(struct pcb_t *caller, // process call
                    int addr, // start address which is aligned to pagesz
                    int pgnum, // num of mapping page
          struct framephy_struct *frames,// list of the mapped frames
             struct vm_rg_struct *ret_rg)// return mapped region, the real mapped fp
{                                        // no guarantee all given pages are mapped
```

```
  //uint32_t * pte = malloc(sizeof(uint32_t));
  struct framephy_struct *fpit = malloc(sizeof(struct framephy_struct));
  //int  fpn;
  int pgit = 0;
  int pgn = PAGING_PGN(addr);

  ret_rg->rg_end = ret_rg->rg_start = addr; // at least the very first space is usable

  fpit->fp_next = frames; //iterator

  /* TODO map range of frame to address space
   *      [addr to addr + pgnum*PAGING_PAGESZ
   *      in page table caller->mm->pgd[]
   */
  /*
  * dựa vào PAGING_PGN(x) để lấy thứ tự trong page table -> pagetable[i] = fpit->fpn | một số thông tin khác
  * usernumber thì ko biết ghi gì ??
  * một số thông số khác thì tùy chỉnh
  */
  for(pgit = 0; pgit < pgnum; pgit++){
    fpit = fpit->fp_next;
    pgn = PAGING_PGN(addr + pgit*PAGING_PAGESZ); //this is index of this page in process's page table
    if(fpit){
      pte_set_fpn(&(caller->mm->pgd[pgn]),fpit->fpn);
      /* Tracking for later page replacement activities (if needed)
       * Enqueue new usage page */
      enlist_pgn_node(&caller->mm->fifo_pgn, pgn);
    }
  }
  ret_rg->rg_end += (pgit-1)*PAGING_PAGESZ;
  return 0;
}
```

+ alloc_pages_range :

```
int alloc_pages_range(struct pcb_t *caller, int req_pgnum, struct framephy_struct** frm_lst)
{
  int pgit, fpn;
  struct framephy_struct* newfp_str;

  /*TODO*/
  for(pgit = 0; pgit < req_pgnum; pgit++)
  {
    if(MEMPHY_get_freefp(caller->mram, &fpn) == 0)
    {
        newfp_str = (struct framephy_struct*)malloc(sizeof(struct framephy_struct));
        newfp_str->fpn = fpn;
        newfp_str->owner = caller->mm;
        if (!*frm_lst)
          *frm_lst = newfp_str;
        else
        {
           newfp_str->fp_next = *frm_lst;
           *frm_lst = newfp_str;
        }

        newfp_str->fp_next = caller->mram->used_fp_list;
        caller->mram->used_fp_list = newfp_str;
    }
    else // ERROR CODE of obtaining somes but not enough frames
    {
        int victim_fpn, victim_pgn, victim_pte;
        int swpfpn = -1;
        if (find_victim_page(caller->mm, &victim_pgn) < 0)
           return -1;
        victim_pte = caller->mm->pgd[victim_pgn];
        victim_fpn = PAGING_FPN(victim_pte);
```

```
    //======================
    newfp_str = (struct framephy_struct*)malloc(sizeof(struct framephy_struct));
    newfp_str->fpn = victim_fpn;
    newfp_str->owner = caller->mm;

    if (!*frm_lst)
      *frm_lst = newfp_str;
    else
    {
       newfp_str->fp_next = *frm_lst;
       *frm_lst = newfp_str;
    }
    //======================
    int i = 0;
    if (MEMPHY_get_freefp(caller->active_mswp, &swpfpn) == 0)
    {
       __swap_cp_page(caller->mram, victim_fpn, caller->active_mswp, swpfpn); // copy content on victim frame to swap memory space
       struct memphy_struct* mswp = (struct memphy_struct*)caller->mswp;
       for (i = 0; i < PAGING_MAX_MMSWP; i++)
       {
          if (mswp + i == caller->active_mswp)
            break;
       }
    }
    else
    {
       struct memphy_struct* mswp = (struct memphy_struct*)caller->mswp;
       swpfpn = -1;
       for (i = 0; i < PAGING_MAX_MMSWP; i++)
       {
          if (MEMPHY_get_freefp(mswp + i, &swpfpn) == 0)
          {
             __swap_cp_page(caller->mram, victim_fpn, mswp + i, swpfpn); // copy content on victim frame to swap memory space
             break;
          }
       }
    }

    if (swpfpn == -1)
       return -3000;

    pte_set_swap(&caller->mm->pgd[victim_pgn], i, swpfpn);
  }
}
```

3.2.3 Test

- Input : os_1_mlq_paging_small_1K

```
2 4 8
2048 16777216 0 0 0
1 p0s  130
2 s3  39
4 m1s  15
6 s2  120
7 m0s  120
9 p1s  15
11 s0 38
16 s1 0
```

Time slice = 2, Number of CPUs = 4, Number of process = 8

- Process :

p0s	s3	m1s	s2	m0s	p1s	s0	s1
1 10 calc alloc 300 0 alloc 300 4 free 0 alloc 100 1 write 100 1	7 11 calc calc calc calc calc calc calc calc calc calc	1 8 alloc 300 0 alloc 100 1 free 0 alloc 100 2 free 2 free 1	20 12 calc calc calc calc calc calc calc calc calc calc	1 7 alloc 300 0 alloc 100 1 free 0 alloc 100 2 write 102 1 20	1 10 calc calc calc calc calc calc calc calc calc calc	12 15 calc calc calc calc calc calc calc calc calc calc	20 7 calc calc calc calc calc calc calc calc

20 read 1 20 20 write 102 2 20 read 2 20 20 write 103 3 20 read 3 20 20 calc free 4 calc	calc calc calc calc calc calc calc		calc calc calc	write 1 2 1000	calc	calc calc calc calc calc	

- Output: ./os os_1_mlq_paging_small_1K

```
ld_routine
Time slot   0
Time slot   1
        Loaded a process at input/proc/p0s, PID: 1 PRIO: 130
        CPU 1: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/s3, PID: 2 PRIO: 39
        CPU 2: Dispatched process  2
print_pgtbl: 0 - 512
00000000: 80000001
00000004: 80000000
Memory content-[pos, content]:
Time slot   3
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  1
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]:
Time slot   4
        Loaded a process at input/proc/m1s, PID: 3 PRIO: 15
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  3
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Memory content-[pos, content]:
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]:
        CPU 3: Dispatched process  2
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Time slot   5
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  1
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]:
Memory content-[pos, content]:
Time slot   6
        Loaded a process at input/proc/s2, PID: 4 PRIO: 120
        CPU 3: Put process  2 to run queue
        CPU 3: Dispatched process  2
        CPU 0: Dispatched process  4
write region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]:
        CPU 2: Put process  3 to run queue
        CPU 2: Dispatched process  3
```

```
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Memory content-[pos, content]: [276, 100]
Time slot   7
        Loaded a process at input/proc/m0s, PID: 5 PRIO: 120
print_pgtbl: 0 - 512
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  5
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Memory content-[pos, content]: [276, 100]
00000000: 80000005
00000004: 80000004
Memory content-[pos, content]: [276, 100]
Time slot   8
        CPU 3: Put process  2 to run queue
        CPU 3: Dispatched process  1
read region=1 offset=20 value=100
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]: [276, 100]
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
        CPU 2: Put process  3 to run queue
        CPU 2: Dispatched process  4
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Memory content-[pos, content]: [276, 100]
Time slot   9
        Loaded a process at input/proc/p1s, PID: 6 PRIO: 15
        CPU 1: Put process  5 to run queue
        CPU 1: Dispatched process  3
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Memory content-[pos, content]: [276, 100]
write region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]: [276, 100]
Time slot  10
        CPU 2: Put process  4 to run queue
        CPU 2: Dispatched process  6
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Memory content-[pos, content]: [276, 102]
        CPU 3: Put process  1 to run queue
        CPU 3: Dispatched process  5
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Memory content-[pos, content]: [276, 102]
```

```
Time slot  11
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  4
        Loaded a process at input/proc/s0, PID: 7 PRIO: 38
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Memory content-[pos, content]: [276, 102]
Time slot  12
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  1
read region=2 offset=20 value=102
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]: [276, 102]
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Memory content-[pos, content]: [276, 102]
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  6
Time slot  13
write region=3 offset=20 value=103
print_pgtbl: 0 - 1024
00000000: 80000001
00000004: 80000000
00000008: 80000003
00000012: 80000002
Memory content-[pos, content]: [276, 102]
print_pgtbl: 0 - 512
00000000: 80000005
00000004: 80000004
Memory content-[pos, content]: [276, 103]
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  7
Time slot  14
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  2
        CPU 2: Processed  1 has finished
        CPU 2: Dispatched process  5
write region=1 offset=20 value=102
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Memory content-[pos, content]: [276, 103]
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  4
Time slot  15
        CPU 3: Processed  2 has finished
        CPU 3: Dispatched process  6
write region=2 offset=1000 value=1
print_pgtbl: 0 - 512
00000000: 80000007
00000004: 80000006
Memory content-[pos, content]: [276, 103][1600, 102]
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  16
        CPU 0: Put process  4 to run queue
```

```
        CPU 0: Dispatched process  4
        CPU 2: Put process  5 to run queue
        CPU 2: Dispatched process  5
write region=0 offset=0 value=0
print_pgtbl: 0 - 512
00000000: 40000000
00000004: 80000006
Memory content-[pos, content]: [276, 103][1600, 102][2024, 1]
        Loaded a process at input/proc/s1, PID: 8 PRIO: 0
Time slot  17
        CPU 2: Processed  5 has finished
        CPU 2: Dispatched process  8
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot  18
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  19
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
Time slot  20
        CPU 0: Processed  4 has finished
        CPU 0 stopped
Time slot  21
        CPU 3: Processed  6 has finished
        CPU 3 stopped
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  22
Time slot  23
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
Time slot  24
        CPU 2: Processed  8 has finished
        CPU 2 stopped
Time slot  25
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  26
Time slot  27
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  28
        CPU 1: Processed  7 has finished
        CPU 1 stopped
```

CHECKLIST We leverage some materials in guidelines, but as usual, this section has a lot of material intensively. Try to keep it not too long

Checklists	Y/N
What was done? Address question such as: With the available resource, we design this, implement that With the meeting, we minus/refine the work target Describe the activities supported under the project Describe the timeline or scheduling	
What was learned about the implementation and management of the projects' activities?	
Design and implementation (3-4 pages, try to keep it less than 6 pages)	
Simulation, testing or evaluation (3-4 pages should be balance with the previous section) Does this experiment consist of objective statements? Double check.	

Did it determine participants' role and contributions Who is coordinator, design the plan, integrated need to verify and design test/evaluation method Who is the architecture design should provide proof-of-concept or at least prototype working code Who is the main developer to implement the details and fix bug Who is quality analyst design the sanity test, the module test or self-test and test scenario	

3.3 Synchronization

3.3.1 Design

Khi hệ thống thực thi với nhiều CPU, các tài nguyên bị tranh chấp cần phải được bảo vệ nhằm tránh những hiện tượng không mong muốn. Trong hệ điều hành đơn giản của bài tập lớn này, mutex lock được sử dụng để bảo vệ tài nguyên khi thực thi chương trình.

3.3.2 Implementation

- get_mlq_proc :

```
struct pcb_t * get_mlq_proc(void)
{
        /*TODO: get a process from PRIORITY [ready_queue].
         * Remember to use lock to protect the queue.
         * */

        pthread_mutex_lock(&queue_lock);
        struct pcb_t * proc = NULL;
        int isEmpty = 0; // value for the second check, if mlq stills empty, then return null
        for (int i = currPrio; i < MAX_PRIO; i++)
        {
                currPrio = i;
                if (mlq_ready_queue[i].size > 0)
                {
                        mlq_ready_queue[i].run += time_slot;

                        //if (mlq_ready_queue[i].run <= mlq_ready_queue[i].maxRun)
                        proc = dequeue(&mlq_ready_queue[i]);

                        if (mlq_ready_queue[i].run >= mlq_ready_queue[i].maxRun || empty(&mlq_ready_queue[i]))
                        {
                                mlq_ready_queue[i].run = 0;
                                currPrio++;
                                if (currPrio >= MAX_PRIO - 1)
                                        currPrio = 0;
                        }

                        break;
                }

                if (currPrio >= MAX_PRIO - 1)
                {
                        currPrio = 0;

                        if (isEmpty)
                                break;

                        i = -1;
                        isEmpty = 1;
                }
        }
        pthread_mutex_unlock(&queue_lock);
        return proc;
}
```

- put_mlq_proc / add_mlq_proc :

```
void put_mlq_proc(struct pcb_t * proc) {
        pthread_mutex_lock(&queue_lock);
        enqueue(&mlq_ready_queue[proc->prio], proc);
        pthread_mutex_unlock(&queue_lock);
}

void add_mlq_proc(struct pcb_t * proc) {
        pthread_mutex_lock(&queue_lock);
        enqueue(&mlq_ready_queue[proc->prio], proc);
        pthread_mutex_unlock(&queue_lock);
}
```

- MEMPHY_get_freefp :

```c
int MEMPHY_get_freefp(struct memphy_struct *mp, int *retfpn)
{
    pthread_mutex_lock(&mem_lock);

    struct framephy_struct *fp = mp->free_fp_list;

    if (fp == NULL)
      return -1;

    *retfpn = fp->fpn;
    mp->free_fp_list = fp->fp_next;

    /* MEMPHY is iteratively used up until its exhausted
     * No garbage collector acting then it not been released
     */
    free(fp);

    pthread_mutex_unlock(&mem_lock);

    return 0;
}
```

- MEMPHY_put_freefp :

```c
int MEMPHY_put_freefp(struct memphy_struct *mp, int fpn)
{
    pthread_mutex_lock(&mem_lock);

    struct framephy_struct *fp = mp->free_fp_list;
    struct framephy_struct *newnode = malloc(sizeof(struct framephy_struct));

    /* Create new node with value fpn */
    newnode->fpn = fpn;
    newnode->fp_next = fp;
    mp->free_fp_list = newnode;

    pthread_mutex_unlock(&mem_lock);

    return 0;
}
```

4 PARTICIPANT AND PROJECT OUTPUTS

4.1 Role and contribution of each member 1

4.1.1 Vai trò

4.1.2 Đóng góp

We leverage some materials in guidelines, but as usual, this section has a lot of material intensively. Try to keep it not too long

Checklists	Y/N
What was done? Address question such as: With the available resource, we design this, implement that With the meeting, we minus/refine the work target Describe the activities supported under the project Describe the timeline or scheduling	
What was learned about the implementation and management of the projects' activities?	
Design and implementation (3-4 pages, try to keep it less than 6 pages)	
Simulation, testing or evaluation (3-4 pages should be balance with the previous section) Does this experiment consist of objective statements? Double check.	

Did it determine participants' role and contributions Who is coordinator, design the plan, integrated need to verify and design test/evaluation method Who is the architecture design should provide proof-of-concept or at least prototype working code Who is the main developer to implement the details and fix bug Who is quality analyst design the sanity test, the module test or self-test and test scenario	

4.2 Project output

4.2.1 Answer Question

4.2.1.1 Scheduling

1. What is the advantage of using priority queue in comparison with other scheduling algorithms you have learned?

- So với các giải thuật khác, priority queue dựa trên độ ưu tiên để sắp xếp các tác vụ. Điều này cho phép các process với độ ưu tiên cao sẽ được xử lý sớm hơn các process với độ ưu tiên thấp, giúp tăng độ tương tác của hệ điều hành với người dùng.
- Đồng thời do priority queue sử dụng định thời ưu tiên (preemptive scheduling) nên cho phép các process có độ ưu tiên cao được xử lý dù các process có độ ưu tiên thấp hơn đang được thực thi.

4.2.1.2 Memory management

1. In this simple OS, we implement a design of multiple memory segments or memory areas in source code declaration. What is the advantage of the proposed design of multiple segments?

CPU bus	PAGE size	PAGE bit	No pg entry	PAGE Entry sz	PAGE TBL	OFFSET bit	PGT mem	MEMPHY	fram bit
20	256B	12	~4000	4byte	16KB	8	2MB	1MB	12
22	256B	14	~16000	4byte	64KB	8	8MB	1MB	12
22	512B	13	~8000	4byte	32KB	9	4MB	1MB	11
22	512B	13	~8000	4byte	32KB	9	4MB	128kB	8
16	512B	8	256	4byte	1kB	9	128K	128kB	4

- Tối ưu hoá sử dụng bộ nhớ, tránh gom thành 1 segment quá lớn
- Các segment có thể khác nhau về kích thước, linh hoạt hơn khi thao tác với bộ nhớ như cấp phát, hay thu hồi
- Các segment có thể ánh xạ sang bộ nhớ thực với địa chỉ khác nhau, giúp linh động về vị trí trong bộ nhớ, không bị giới hạn bởi một khối trong bộ nhớ thực
- Giảm hiện tượng phân mảnh nội.

2. What will happen if we divide the address to more than 2 levels in the paging memory management system?

- Với các hệ thống phân trang từ hai cấp trở lên, ta gọi là hệ thống phân trang đa cấp. Ở hệ thống phân trang đa cấp, các vùng địa chỉ sẽ ánh xạ đến những địa chỉ ở vùng địa chỉ cấp thấp hơn, riêng vùng địa chỉ cuối cùng sẽ ánh xạ đến vùng nhớ tương ứng trong bộ nhớ thực.
- Ví dụ :
  Một kiến trúc 32-bit sẽ có 2^{32} byte cần được truy cập tương ứng 2^{32} địa chỉ. Giả sử 1 trang có kích thước là 4KB = 2^{12} byte.

Với hệ thống phân trang một cấp, bộ nhớ cần lưu $2^{32}/2^{12} = 2^{20}$ trang, tương ứng với 2^{20} entry, mỗi entry là 4 byte nên bộ nhớ cần lưu bảng phân trang có kích thước 4.2^{20} byte = 4Mb dẫn đến lãng phí bộ nhớ chỉ dành cho việc lưu trữ trang.
Để giải quyết vấn đề này, ta sử dụng bảng phân trang đa cấp vì chỉ cần vùng nhớ cấp 1 được lưu vào bộ nhớ và các trang khác sẽ được tải lên khi cần thiết giúp tiết kiệm dung lượng cho bộ nhớ.

- Ưu điểm :
+ Giảm bộ nhớ cần thiết để lưu trữ bảng phân trang
+ Tra cứu bảng phân trang nhanh hơn vì số entry mỗi cấp nhỏ dần khi cấp tăng lên
+ Giúp bộ nhớ linh hoạt hơn khi có thay đổi về tổ chức không gian bộ nhớ
- Nhược điểm :
+ Tăng độ phức tạp khi có càng nhiều cấp
+ Tăng chi phí để truy cập đến bảng trang cần tìm do mỗi cấp phải được duyệt để tìm entry tương ứng.
+ Phân mảnh bộ nhớ do các entry có thể không liên tục, làm tăng chi phí khi truy cập bộ nhớ

3. What is the advantage and disadvantage of segmentation with paging?

- Segmentation with paging là một kỹ thuật quản lý bộ nhớ kết hợp những điểm mạnh của hai kỹ thuật segmentation và paging lại với nhau bằng cách chia không gian bộ nhớ thành các segment, và chia các segment thành các page giúp việc cấp phát vùng nhớ trở nên linh hoạt hơn vì mỗi segment có thể có kích thước khác nhau.
- Ưu điểm :
+ Các segment có kích thước tùy ý, giúp cho việc cấp phát linh hoạt hơn đồng thời giảm hiện tượng phân mảnh ngoại.
+ Tối ưu hoá sử dụng bộ nhớ và hiệu năng của hệ thống vì không cần load tất cả page. Segmentation with paging chỉ load những page cần thực thi, từ đó giảm thời gian thực hiện swap và xử lý được nhiều process cùng lúc.
- Nhược điểm :
+ Việc hiện thực rất phức tạp.
+ Quản lý nhiều segment và page làm tiêu tốn tài nguyên CPU vì có thể có nhiều page được lưu trong RAM nhưng chỉ có một page hoạt động trong thời điểm nhất định.
+ Page có kích thước cố định nên vẫn còn hiện tượng phân mảnh nội.

4.2.1.3 Memory management

1. What will happen if the synchronization is not handled in your simple OS? Illustrate by example the problem of your simple OS if you have any.
Khi hiện thực một Hệ điều hành, nếu không xử lý đồng bộ, các tài nguyên dùng chung bởi nhiều luồng CPU sẽ dẫn đến hiện tượng race condition làm cho dữ liệu sinh ra lỗi và logic chương trình không còn đúng.

4.2.2 Main output

Mô phỏng được một hệ điều hành đơn giản với tài nguyên và các process được định nghĩa trước

- Scheduler : thiết kế và hiện thực giải thuật Multilevel queue scheduling
- Memory management : thiết kế và hiện thực bộ nhớ ảo, bộ nhớ vật lý và ánh xạ dựa trên kỹ thuật phân trang

4.2.3 Achievement

- Scheduler : Nhóm có thêm kiến thức về cách hiện thực giải thuật định thời Scheduling, tìm hiểu về ưu điểm và nhược điểm của giải thuật so với các giải thuật định thời khác.
- Memory management : Nhóm có thêm kiến thức về nguyên lý hoạt động của kỹ thuật phân trang, về cách hiện thực bộ nhớ thực, bộ nhớ ảo và phương thức ánh xạ giữa hai bộ nhớ.

Checklists	Y/N

Directly achievable product of project's completed activities Standalone standards: Protocol, rule, procedure, algorithm, guidelines Software product: program/application, libraries, APIs	
Verify the matching of these outputs with the project's completed activities? Is it an over claim? Is there any completed activities without nothing (output), if yes, re-evaluate: we did it unfinished or / lost focus on the objectives or problematic planning?	
What were the **main output** of the project? Identify any output that were planned but which has not materialised Specify when these output will be completed, if it falls into out-of-scope including plan for any future publications (if possible)	
What were the main specific **achievements** or **practice influence** What was learned about the production or realisation of the project. What contributed to these output and what lessons you may draw from your experiences	
If appropriate, highlight any unique or innovative outputs If appropriate, explain why output were not completed or were poor of quality	

4.3 Project outcome

Verify which outcomes we achieve through this project (?)

L.O.1	**Describe on how to apply fundamental knowledge of computing and mathematics in an operating system**

L.O.1.1	Define the functionality and structures that a modern operating system must deliver to meet a particular need.
L.O.1.2	Explain virtual memory and its realisation in hardware and software.
L.O.2	**Able to report the tradeoffs between the performance and the resource and technology constraints in a design of an operating system.**
L.O.2.1	Compare and contrast common algorithms used for both preemptive and non-preemptive scheduling of tasks in operating systems
L.O.2.2	Compare and contrast different approaches to file organisation, recognizing the strengths and weaknesses of each.
L.O.3	**Describe main operating system concepts and their aspects that are useful to realise concurrent systems and describe the benefits of each.**
L.O.3.1	Compare and contrast different methods for process synchronisation.

- Nhóm đã thiết kế thành công một hệ điều hành đơn giản gồm hai thành phần chính : **Scheduler** và **Paging-based Memory Management** nhằm sắp xếp và quản lý tài nguyên bộ nhớ khi thực thi chương trình
- Trong bài tập lớn này, **Scheduler** được thiết kế theo hai thuật toán định thời (preemptive) là Multi-level queue để thể hiện cơ chế priority và Round Robin để giảm thời gian phản hồi của các process.
- Về **Paging-based Memory Management**, để truy cập vào các frame được sắp xếp không theo thứ tự trong bộ nhớ vật lý không liên tục (non-contiguous), nhóm sử dụng các page trong virtual memory để ánh xạ và truy cập các frame đó.
- Về **xử lý đồng bộ**, nhóm sử dụng mutex_lock và conditional_lock để khoá các tài nguyên nhằm tránh xảy ra tranh chấp giữa các process.

ANNEXES

1. Terms of Reference

[1] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne, (2018), ” Operating System Concept”, Tenth Edition